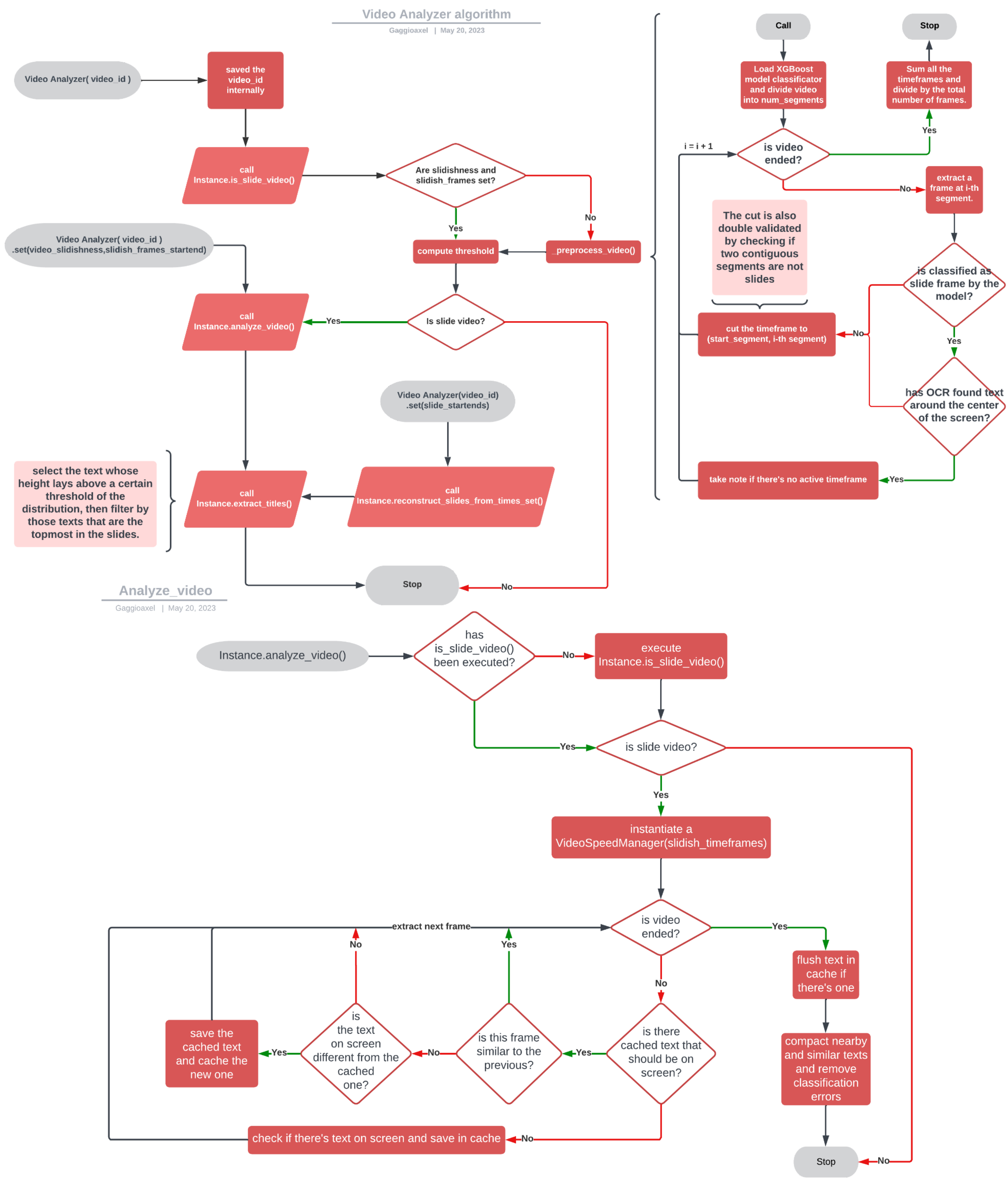
EDURELL Project

Video Segmentation

Algorithms



Implementation

The coarse-analysis of the video that finds the percentage of slides in the video is calculated by using a pre-trained ML model that recognizes the “slidish” images and is double checked with the OpenCV library:

Algorithm 1 Video Pre-Processing

F: video frames,
OUT list of cut frames’ windows
N: number of video frames,
M: number of segments
$Model$: XGBoost pre-trained ML model that recognizes slides in images
procedure Cut Non-Slide Frames(M=150)
    for $i \in \{2 \dots 150\}$ **do**
        if both $(F_{(i-1)*N/M}, F_{i*N/M})$ not IsSlideFrame($Model$) and $StartFrame$ is set **then**
            $OUT_j \leftarrow (StartFrame, i-1)$
            unset $StartFrame$
        else if $F_{i*N/M}$ IsSlideFrame($Model$) and $StartFrame$ is not set **then**
            if ValidatedWithOCR($F_{i*N/M}[\frac{1}{3}(width, height) <-> \frac{2}{3}(width, height)]$) **then**
                $StartFrame \leftarrow i-1$
    return OUT

Then if the video has been classified as “slidish enough” the video is analyzed based on this algorithm:

Algorithm 2 Text From Video Segmentation

OUT: a list of *already processed* slide frames
CT: *currently-on-screen text*
V: video reference
procedure Video Segmentation
    for every frame F_i in V **do**
        if there’s no CT and F_i contains some $text$ **then**
            $CT \leftarrow (text,$ FirstFrameOccurenceOfThisText$(V, text))$
        else if there’s some CT and F_i is different enough[12] from F_{i-1} **then**
            if CT and $text$ extracted from F_i are not the same **then**
                $OUT_j \leftarrow (CT,$ LastFrameOccurenceOfThisText$(V, CT.text))$
    return OUT

Classes description

Video Segmentation in the Edurell platform is performed in Python by using these main classes:

In the image.py file:

- ImageClassifier (IC): an image wrapper that finds faces and text in the image and manages color scheme conversions

In the video.py file:

- LocalVideo (LV): class that manages OpenCV video file loading, frame cursor set, frames extraction, conversion and resize.

- VideoSpeedManager (VSM): wrapper of LocalVideo that manages the logic of frames extraction.

In the segmentation.py file:

- TimedAndFramedText (TFT): dataclass that contains the following informations of the slide segment of the video:
    - Full text of that slide
    - X and Y positions and Width and Height (normalized) of the bounding boxes of every sentence indexed from the full text
    - Initial and last frame number of the video where the text appear on screen

  With some utility function that allow to insert multiple start-end windows of frames that contain that text:

```python
def extend_frames(self, other_start_end_list:List[Tuple[(int,int)]]) -> None:
    '''
    extend this element's start_end frames with the other list in a sorted way
    '''
    for other_start_end_elem in other_start_end_list:
        insort_left(self.start_end_frames,other_start_end_elem)

def get_full_text(self):
    return self._full_text

def get_split_text(self):
    '''
    returns the full text splitted by new lines without removing them
    '''
    return self._full_text.split("(?<=\n)")

def get_framed_sentences(self):
    '''
    converts the full text string into split segments of text with their respective bounding boxes
    '''
    full_text = self._full_text
    return [((full_text[start_char_pos:end_char_pos]),bb) for (start_char_pos,end_char_pos),bb in self._framed_sentences]

def merge_adjacent_startend_frames(self,max_dist:int=15) -> 'TimedAndFramedText':
    '''
    Merges this object adjacent (within a max_dist) frame times
    '''
    start_end_frames = self.start_end_frames
    merged_start_end_frames = []
    curr_start,curr_end = start_end_frames[0]
    for new_start,new_end in start_end_frames:
        if new_start-curr_end <= max_dist:
            curr_end = max(curr_end,new_end)
        else:
            merged_start_end_frames.append((curr_start,curr_end))
            curr_start = new_start
            curr_end = new_end
    merged_start_end_frames.append((curr_start,curr_end))
    self.start_end_frames = merged_start_end_frames
    return self
```

- VideoAnalyzer (VA): class that contains the logic to read a video and extract from it:
    - The transcript, and its segmentation into timed sentences
    - The keyframes (based on the previous segmentation method which is based on colour histograms)

```python
def _create_keyframes(self,start_times,end_times,S,seconds_range, image_scale:float=1,create_thumbnails=True):...

def get_transcript(self,lang:str='en'):...

def transcript_segmentation(self, subtitles, c_threshold=0.22, sec_min=35, S=1, frame_range=15,create_thumbnails=True):...
```

- The percentage of slide frames over the entire video length, classification based on a threshold

```python
_preprocess_video(self, vsm:VideoSpeedManager,num_segments:int=150,estimate_threshold=False,_show_info=False):
'''
Split the video into `num_segments` windows frames, for every segment it's taken the frame that's far enough to guarantee
The current frame is analyzed by XGBoost model to recognize the scene\n
If there are two non-slide frames consecutively the resulting frame window is cut\n
Bounds are both upper and lower inclusive to avoid a miss as much as possible\n
Then both are compared in terms of cosine distance of their histograms (it's faster than flattening and computing on pure
Lastly the distance between wach frame is selected as either the average of values, either with fixed value.\n
In this instance with videos that are mostly static, the threshold is set to 0.9999
#TODO further improvements: for more accuracy this algorithm could include frames similarity to classify a segment as sli

Returns
----------
The cosine similarity threshold and the list of frames to analyze (tuples of starting and ending frames)

----------
Example
----------
A video split into 10 segments:\n\n
slide_segments : 0,1,3,6,9,10\n
non_slide_segments : 2,4,5,7,8\n
results in segmentation = [(0,4),(5,7)(8,10)]\n
with holes between segments 4-5 and 7-8
'''
num_frames = vsm.get_video().get_count_frames()
speed = floor(num_frames / (num_segments))
vsm.lock_speed(speed)
iterations_counter:int = 0
txt_cleaner = TextCleaner()
if estimate_threshold:
    cos_sim_values = empty((num_segments,vsm.get_video().get_dim_frame()[2]))
    #dists = empty((num_segments,vsm.get_video().get_dim_frame()[2]))

# Optimization is performed by doing a first coarse-grained analysis with the XGBoost model predictor
# then set those windows inside the VideoSpeedManager
scene_model = XGBoostModelAdapter(os.path.dirname(os.path.realpath(__file__))+"/xgboost/model/xgboost500.sav")

start_frame_num = None
frames_to_analyze:List[Tuple[int,int]] = []
answ_queue = deque([False,False])
curr_frame = ImageClassifier(image_and_scheme=[None,vsm._color_scheme])
prev_frame = curr_frame.copy()
frame_w,frame_h,num_colors = vsm.get_video().get_dim_frame()
while iterations_counter < num_segments:
    prev_frame.set_img(vsm.get_frame())
    curr_frame.set_img(vsm.get_following_frame())
    if scene_model.is_enough_slidish_like(prev_frame):
        frame = prev_frame.get_img()
        # validate slide in frame by slicing the image in a region that removes logos (that are usually in corners)
        region = (slice(int(frame_h/4),int(frame_h*3/4)),slice(int(frame_w/4),int(frame_w*3/4)))
        prev_frame.set_img(frame[region])
        # double checks the text
        is_slide = bool(txt_cleaner.clean_text(prev_frame.extract_text(return_text=True)).strip())
    else:
        is_slide = False
    answ_queue.appendleft(is_slide); answ_queue.pop()

    # if there's more than 1 True discontinuity -> cut the video
    if any(answ_queue) and start_frame_num is None:
        start_frame_num = int(clip(iterations_counter-1,0,num_segments))*speed
    elif not any(answ_queue) and start_frame_num is not None:
        frames_to_analyze.append((start_frame_num,(iterations_counter-1)*speed))
        start_frame_num = None

    if estimate_threshold:
        cos_sim_values[iterations_counter,:] = prev_frame.get_cosine_similarity(curr_frame)
        #dists[iterations_counter,:] = curr_frame.get_mean_distance(prev_frame)
    iterations_counter+=1
    #print(answ_queue[0]);plt.imshow(curr_frame.get_img(),cmap='gray');plt.show()
    #print(f" Estimating cosine_similarity_threshold: {ceil((iterations_counter)/num_segments * 100)}%",end='\r')
    if _show_info: print(f" Coarse-grained analysis: {ceil((iterations_counter)/num_segments * 100)}%",end='\r')
if start_frame_num is not None:
    frames_to_analyze.append((start_frame_num,num_frames-1))

if estimate_threshold:
    cos_sim_img_threshold = clip(average(cos_sim_values,axis=0)+var(cos_sim_values,axis=0)/2,0.9,0.9999)
    #cos_sim_img_threshold = clip(cos_sim_values.min(axis=0),0.9,0.99999)
    #   can't estimate correctly the cosine similarity threshold with average, too dependant from the segments chosen and
    #   neither can set to max because it's always more than 1 neither to min because it's too low
    #diff_threshold = average(diffs,axis=0)+3*var(diffs,axis=0)
    #dist_threshold = (dists.max(axis=0) - dists.min(axis=0)) / 2
else:
    cos_sim_img_threshold = ones((1,num_colors))*0.9999

if _show_info:
    if estimate_threshold:
        print(f"Estimated cosine similarity threshold: {cos_sim_img_threshold}")
    else:
        print(f"Cosine_similarity threshold: {cos_sim_img_threshold}")
        #print(f"Estimated mean_dist_threshold: {dist_threshold}")
    print(f"Frames to analyze: {frames_to_analyze} of {num_frames} total frames")
self._video_slidishness = sum([frame_window[1] - frame_window[0] for frame_window in frames_to_analyze])/(num_frames-1)
self._cos_sim_img_threshold = cos_sim_img_threshold
self._frames_to_analyze = frames_to_analyze #dist_threshold, frames_to_analyze
```

- The slide frames are extracted by analyzing the whole video

```python
def analyze_video(self,color_scheme_for_analysis:int=COLOR_BGR,_show_info:bool=False,_plot_contours=False):
    '''
    Firstly the video is analized in coarse-grained way, by a ML model to recognize frames of slides
    and the threshold for the difference between two frames is concurrently estimated. \n
    The method uses an ImageClassifier to detect text in the frames of the video
    Then saves that text in a collision stack.\n
    It then looks for changes in the collision stack with respect to the text in the current frame
    flushes the differences in the output list.\n
    Finally, it calls two helper methods to clean up the output list by combining partial words and merging adjacent frames.

    Parameters :
    ------------
    - color_scheme_for_analysis : is a predefined value that must be either COLOR_BGR or COLOR_RGB from image.py
    - _show_info : prints in stdout progression and set thresholds
    - _plot_contours : shows images with bounding boxes once the video has been analyzed

    Returns :
    ---------
    None but sets internal text that can be retrieved with ``get_extracted_text()``
    '''
```

Then each segment of the output list is compacted by merging similar texts and contiguous segments of same text. Lastly each section is validated with a double check for each segment

- Slide's titles are chosen with statistical analysis on the height of the text and it's position with respect to the other text of the slide:

```python
def extract_titles(self,quant:float=.8,axis_for_outliers_detection:Literal['w','h','wh']='h',union=True,with_times=True) -> list:
    """
    Titles are extracted by performing statistics on the axis defined, computing a threshold based on quantiles\n
    and merging results based on union of results or intersection\n
    #TODO improvements: now the analysis is performed on the whole list of sentences, but can be performed:\n
        - per slide\n
        - then overall\n
    but i don't know if can be beneficial, depends on style of presentation.
    For now the assumption is that there's uniform text across all the slides and titles are generally bigger than the other text

    Then titles are further more filtered by choosing that text whose size is above the threshold, only if it's
    the first sentence of the slide\n

    Prerequisites :
    ------------
    Must have runned analyze_video()

    Parameters :
    ----------
    - quant : quantile as lower bound for outlier detection
    - axis_for_outliers_detection : axis considered for analysis are 'width' or 'height' or both
    - union : lastly it's perfomed a union of results (logical OR) or an intersection (logical AND)
    - with_times : if with_times returns a list of tuples(startend_frames, text, bounding_box)
            otherwise startend_frames are omitted

    Returns :
    ---------
    List of all the detected titles
    """
    assert axis_for_outliers_detection in {'w','h','wh'} and 0 < quant < 1 and self.get_extracted_text(format='list[text,time,box]') is not None
    # convert input into columns slice for analysis
    sliced_columns = {'w':slice(2,3),'h':slice(3,4),'wh':slice(2,4)}[axis_for_outliers_detection]
    texts_with_bb = self.get_extracted_text(format='list[text,time,box]')

    # select columns
    axis_array = array([text_with_bb[2][sliced_columns] for text_with_bb in texts_with_bb],dtype=dtype('float','float'))

    # compute statistical analysis on columns
    indices_above_threshold = list(where((axis_array > quantile(axis_array,quant,axis=0)).any(axis=1))[0]) if union else \
                              list(where((axis_array > quantile(axis_array,quant,axis=0)).all(axis=1))[0])

    slides_group = []
    # group by slide
    # x[0] is one element of indices_above_threshold to compare, x[1] is another
    # [1] accesses the startend seconds of that TimedAndFramedText  [text, startend, bounding_boxes]
    # [0] picks the start second of that object [start_second, end_second]
    for _,g in groupby(enumerate(indices_above_threshold),lambda x: texts_with_bb[x[0]][1][0] - texts_with_bb[x[1]][1][0]):
        slides_group.append(list(reversed(list(map(lambda x:x[1], g)))))
    slides_group = list(reversed(slides_group))

    # remove indices of text that are classified as titles but are just big text that's not at the top of every slide
    for group in slides_group:
        for indx_text in group:
            if indx_text > 0 and indx_text-1 not in group and (                    # if i'm not at the first index and the text of the previous index is not in the group
               texts_with_bb[indx_text-1][1][0] == texts_with_bb[indx_text][1][0] and  # and the text is in the same slide
               texts_with_bb[indx_text-1][2][1] < texts_with_bb[indx_text][2][1]):     # and has a y value lower than my current text (there's another sentence before this one)
                indices_above_threshold.remove(indx_text)

    # selects the texts from the list of texts
    if not with_times:
        return itemgetter(*indices_above_threshold)(list(zip(*list(zip(*texts_with_bb))[0:3:2])))
    return itemgetter(*indices_above_threshold)(texts_with_bb)
```

- Concepts are extracted from the title with phrasemachine and definitions and in-depths search are calculated with an heuristic (the definition could be in a timeframe of a number of seconds around the slide first appearance where the concept is cited in the transcript, and the in-depth could be the whole duration of the slide):

```python
def adjust_or_insert_definitions_and_indepth_times(self,burst_concepts:List[dict],definition_tol_seconds:float = 3,_show_output=False):
    '''
    This is an attempt to find definitions from timed sentences of the transcript and the timed titles of the slides.\n
    Heuristic is that if there's a keyword in the title of a slide (frontpage slide excluded)
    find the first occurence of that keyword in the transcript within a tolerance seconds window before and after the appearance of the slide
    set that as "definition" only if it contains the keyword of the title .\n
    Heuristic for the in-depth is that after definition there's an in-depth of the slide, this means that the concept is explained further there, until the slide with that title won't disappear.\n

    On the algorithmic side sentences of the transcript used by the heuristic are mapped to the conll version of the text,
    cleaning operation is performed to remove errors (it groups the contiguous hit of every used sentence of the transcript to the conll by groups len and picks the biggest)\n
    Then the mapped sentences are aggregated by start and end sentence ids\n
    Lastly if the burst analysis has different definition times it is overwritten, if it does not contain the definition, this is appended at the end
    '''
    if self._slide_titles is None:
        raise Exception('slide titles not set')

    def extract_defs_and_indepths(titles:'list[dict]',timed_sentences:'list[dict]',definition_tol_seconds:float,_show_output=False):...

    # extract definitions and in-depths in the transcript of every title based on slide show time and concept citation (especially with definition)
    # alg is O(kn) with k titles and n sentences of the transcript
    timed_sentences = get_timed_sentences(self.get_transcript()[0],[sent.metadata["text"] for sent in parse(get_text(self._video_id,return_conll=True)[1])])
    video_defs, video_in_depths = extract_defs_and_indepths(self._slide_titles,timed_sentences,definition_tol_seconds,_show_output=_show_output)

    def seconds_to_h_mm_ss_dddddd(time:float):
        millisec = str(time%1)[2:8]
        millisec += '0'*(6-len(millisec))
        seconds = str(int(time)%60)
        seconds = '0'*(2-len(seconds)) + seconds
        minutes = str(int(time/60))
        minutes = '0'*(2-len(minutes)) + minutes
        hours = str(int(time/3600))
        return hours+':'+minutes+':'+seconds+'.'+millisec

    # creating or modifying burst_concept_objects of the video results
    added_concepts = []
    for dict_num,concepts_video_dict in enumerate([video_defs,video_in_depths]):
        concepts_used = {concept:False for concept in concepts_video_dict.keys()}
        concept_description_type = 'Definition' if dict_num == 0 else 'In Depth'
        for burst_concept in burst_concepts:
            if burst_concept["concept"] in concepts_video_dict.keys() and burst_concept["description_type"] == concept_description_type:
                if burst_concept["start_sent_id"] != concepts_video_dict[burst_concept["concept"]][0][0] or \
                   burst_concept["end_sent_id"] != concepts_video_dict[burst_concept["concept"]][-1][0]:

                    burst_concept['start_sent_id'] = concepts_video_dict[burst_concept["concept"]][0][0]
                    burst_concept['end_sent_id'] = concepts_video_dict[burst_concept["concept"]][-1][0]
                    burst_concept['start'] = seconds_to_h_mm_ss_dddddd(concepts_video_dict[burst_concept["concept"]][0][1]["start"])
                    burst_concept['end'] = seconds_to_h_mm_ss_dddddd(concepts_video_dict[burst_concept["concept"]][-1][1]["end"])
                    burst_concept['creator'] = "Video_Analysis"
                    concepts_used[burst_concept["concept"]] = True

        for concept_name in concepts_video_dict.keys():
            if not concepts_used[concept_name]:
                burst_concepts.append({ 'concept':concept_name,
                                        'start_sent_id':concepts_video_dict[concept_name][0][0],
                                        'end_sent_id':concepts_video_dict[concept_name][-1][0],
                                        'start':seconds_to_h_mm_ss_dddddd(concepts_video_dict[concept_name][0][1]['start']),
                                        'end':seconds_to_h_mm_ss_dddddd(concepts_video_dict[concept_name][-1][1]["end"]),
                                        'description_type':concept_description_type,
                                        'creator':'Video_Analysis'})
                added_concepts.append(concept_name)

    return added_concepts,burst_concepts
```

- Slides can be reconstructed from the times saved in the database in the form of timeframes:

```python
def reconstruct_slides_from_times_set(self):
    assert self._slide_startends is not None, "Must firstly load (set) startend frames read from database to run this function"
    slide_startends = self._slide_startends
    frame = ImageClassifier(image_and_scheme=[None,COLOR_BGR])
    loc_video = LocalVideo(self._video_id)
    TFT_list = []
    for slide_start_seconds,slide_end_seconds in slide_startends:
        slide_frames_startend = (loc_video.get_num_frame_from_time(slide_start_seconds),loc_video.get_num_frame_from_time(slide_end_seconds))
        loc_video.set_num_frame(slide_frames_startend[0])
        frame.set_img(loc_video.extract_next_frame())
        text_extracted = frame.extract_text(return_text=True,with_contours=True)
        TFT_list.append(TimedAndFramedText(text_extracted,[slide_frames_startend]))
    self._text_in_video = TFT_list
```

- Each step can be a start point by setting the internal variables from the data read from the database:

```python
def set(self,video_slidishness=None,slidish_frames_startend=None,slide_startends=None,titles=None):
    if video_slidishness is not None:
        self._video_slidishness = video_slidishness
    if slidish_frames_startend is not None:
        self._frames_to_analyze = slidish_frames_startend
    if slide_startends is not None:
        self._slide_startends = slide_startends
    if titles is not None:
        self._slide_titles = titles
    return self
```

- ➔ A process scheduler that automatically segmentates the videos in a global queue of segmentations and saves the results on the database:

```python
def _run_jobs(queue):
    '''
    Periodically checks if the segmentation queue is empty, if not it starts a segmentation and runs it until the end\n
    if it's empty it goes to sleep to avoid cpu usage
    '''
    global client
    client = db_mongo.open_new_socket()
    global db
    db = client.edurell
    while True:
        print(" Scheduler says: No jobs",end="\r")
        if len(queue) > 0:
            video_id = queue.pop(0)
            print("Segmentation job on "+video_id+" starts  working...")
            vid_analyzer = VideoAnalyzer(video_id)
            # if there's no data in the database check the video slidishness
            video_slidishness,slide_frames = vid_analyzer.is_slide_video(return_value=True,return_slide_frames = True,_show_info=True)
            # prepare data for upload
            segmentation_data = {'video_id':video_id,'video_slidishness':video_slidishness,'slidish_frames_startend':slide_frames}
            if vid_analyzer.is_slide_video():
                # if it's classified as a slides video analyze it and insert results in the structure that will be uploaded online
                vid_analyzer.analyze_video(_show_info=True)
                results = vid_analyzer.extract_titles()

                #from pprint import pprint
                #pprint(results)

                # insert titles data in the structure that will be loaded on the db
                segmentation_data = {**segmentation_data,
                                     **{'slide_titles':[{'start_end_seconds':start_end_seconds,'text':title,'xywh_normalized':bb} for (title,start_end_seconds,bb) in results],
                                        'slide_startends': vid_analyzer.get_extracted_text(format='set[times]')}}
            db_mongo.insert_video_text_segmentation(segmentation_data)
            print(f"Job on {video_id} done!"+" "*20)
        time.sleep(10)

def workers_queue_scheduler(queue):
    '''
    Creates a separated process that runs the segmentation of every video in the queue
    '''
    Process(target=_run_jobs,args=(queue,)).start()
```